Likelihood Exploration

Instructions

This document will guide you through an exercise designed to deepen your understanding of likelihood and maximum-likelihood estimation of linear regression models.

Before you start, choose roles for each person on your team:

- One person to type in R
- A code expert to help interpret code (someone most confident in R coding)
- A manager - will read instructions out loud and keep the group on task
- Additional roles if you have more group mates:
  - inclusion (try to ensure everyone contributes and is included) and
  - notes reference (have the course notes or your class notes ready for quick reference as needed)

Data

Today we will use a small dataset to try to answer: **What makes cheese tasty?** …while learning about **Maximum Likelihood Estimation**.

The dataset `cheddar` from package `faraway` has data on expert cheese tastiness ratings, as well as several chemical properties of each cheese.

(Unfortunately, we don't have a lot of detail about the units of measure or method of determining these properties – it is obvious that Dr. Faraway, who provided the data, is a statistician and not a chemist.)

Dataset is in the package `faraway`, so all you have to do is load the package with `library()` and then you can start using the `cheddar` dataset immediately.

```r
library(faraway)
glimpse(cheddar)
```

```
Rows: 30
Columns: 4
$ taste  <dbl> 12.3, 20.9, 39.0, 47.9, 5.6, 25.9, 37.3, 21.9, 18.1, 21.0, 34.9~
$ Acetic <dbl> 4.543, 5.159, 5.366, 5.759, 4.663, 5.697, 5.892, 6.078, 4.898, ~
$ H2S    <dbl> 3.135, 5.043, 5.438, 7.496, 3.807, 7.601, 8.726, 7.966, 3.850, ~
$ Lactic <dbl> 0.86, 1.53, 1.57, 1.81, 0.99, 1.09, 1.29, 1.78, 1.29, 1.58, 1.6~
```

Before you continue, make one, *maybe* two, very quick exploratory data plots to see whether you note any trends in `taste` depending on the `H2S`, `Acetic` acid, or `Lactic` acid content of the cheeses in the dataset.

Regression model

There are 30 data points, so we'd be best off considering a model with 1-2 predictors and not all 3. For this exercise, let's do just one predictor; you can choose as a group which one you want to try. (This is "night science" – fun and exploration – so we won't worry about the fact that you are probably choosing which one to use based on the data plots you just made!)

Adjust the code below to fit a one-predictor linear regression model with **your chosen predictor**.

```r
cheese_model <- lm(taste ~ predictor, data = cheddar)
```

```
Error in eval(predvars, data, env): object 'predictor' not found
```

You know how to view the model summary and get the parameter estimates for the intercept and slope:

```r
summary(cheese_model)
```

```
Error in h(simpleError(msg, call)): error in evaluating the argument 'object' in selecting a
```

Once you've gotten this far, check in with the prof to let her know where you are and if you have any questions.

Likelihood, instead

We can use likelihood to compare the goodness-of-fit of two models fitted to the same data. That's part of how AIC and BIC work.

But we can use likelihood another way, too: to "fit" a single model to data, *figuring out which exact slope and intercept values are BEST for a given dataset and model.*

How can we use **likelihood** to find these "best" slope and intercept parameter estimates, *ourselves*?

What you'll do next isn't exactly what R does to fit an `lm()`. For the linear regression case, smart folks have used calculus and linear algebra to derive analytical solutions for the parameter estimates $\hat{\beta}$ in terms of the dataset - that's why `lm()` can be super fast.

And in cases where R does need to maximize a likelihood to fit a model, it uses smarter methods than just trying a huge range of reasonable possibilities for slope and intercept, getting the likelihood of the data for *every* slope/intercept guess, and finally choosing the best among them. (That is what we are about to do.) This way (it's called a grid search) is a great illustration of the *idea* of maximum-likelihood estimation, even if it's not the fastest algorithm to carry it out.

You don't have to be able to replicate the R code in the following sections. But do your best to work as a group to UNDERSTAND what each code chunk is doing. I suggest adding additional explanation or code comments as needed.

Guess the parameters

Based on what you have seen in the scatter plots, you can probably give reasonable upper and lower boundaries on the intercept and slope for your regression line (don't use the estimates from the `lm()` summary to give unreasonably precise guesses - the point here is to give a wider range to "search" for the best parameter values).

Adjust the code below so that your range of reasonable intercepts and slopes is covered (you may change the `from` and `to` values – those are the minimum and maximum values R will try as candidate slope or intercept values).

How can you guess this? Try to guesstimate the slope and intercept based on your data plot. If needed, check your work by CHEATING and looking at R's `lm()summary()` – make sure the correct slope and intercept estimates are within your guessed ranges!

```
MLE_grid <- expand_grid(intercept = seq(from = -15, by = 0.1, to = 10),
                        slope = seq(from = 3, by = 0.1, to = 7))
glimpse(MLE_grid)
```

```
Rows: 10,291
Columns: 2
$ intercept <dbl> -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, ~
$ slope     <dbl> 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, ~
```

Each row of `MLE_grid` give ONE set of guesses of a possible slope and intercept for our regression model.

Now, we want to compute the **likelihood of the dataset given each of those sets of parameters**.

We can compare the likelihoods to see which estimates are best!

Once you've gotten this far, check in with to the prof to let her know where you are and if you have any questions.

Compute the likelihood: Fitted values

How will we compute the likelihood of the dataset, given a particular slope and intercept guess? Well, first we need to get the model *residuals* (since we compare those to a normal distribution in order to get the likelihood).

And to get the residuals, we need to subtract the model-predicted values $\hat{y}$ from the observed values of the response y.

We already have the response variable values y, in the dataset. But we need the predicted values $\hat{y}$! We have to compute them by hand, since we are not using `lm()` – we can't use `predict()` without the fitted model object that comes from `lm()`!

But no problem, we can do this.

Our model is:

$$y = \beta_0 + \beta_1 x + \epsilon$$

And according to this model, the predicted value of y (on average) is $\beta_0 + \beta_1 x$.

Well, x is the predictor – we have data on that. We also have candidate values of β_0 (the intercepts) and β_1 (the slopes).

We will make our own function to compute fitted values, given a dataset and candidate slope and intercept.

The ONLY change you need to make to the code below is to make sure it refers to "your" predictor, not `H2S`.

Running this chunk will create a FUNCTION you can use later – so there will be no output yet.

```r
get_fitted <- function(slope, intercept, data){
  fitted <- intercept + slope * pull(data, H2S)
}

get_fitted <- Vectorize(get_fitted, c('slope', 'intercept'), SIMPLIFY = FALSE)
```

Now we can add a column to our `MLE_grid` with the fitted values for slope/intercept guess.

This will take a while to run.

When it's done, look what has happened: each *row* of the dataset (corresponding to one guessed slope and intercept) contains a *list* of all 30 fitted values! (Cool.)

```r
MLE_grid <- MLE_grid |>
  mutate(fitted = get_fitted(slope, intercept, data = cheddar))
glimpse(MLE_grid)
```

```
Rows: 10,291
Columns: 3
$ intercept <dbl> -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, ~
$ slope     <dbl> 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, ~
$ fitted    <list> <-5.595, 0.129, 1.314, 7.488, -3.579, 7.803, 11.178, 8.898,~
```

Compute the likelihood: Residuals

OK. Shoot. We didn't actually need the fitted values themselves: we need the *residuals*. We can compute them in a similar way, though. We just need to subtract the fitted values from the observed response variable values!

Again make sure you change the code so it refers to YOUR predictor, not `H2S`.

```r
get_resids <- function(slope, intercept, data){
  fitted <- intercept + slope * pull(data, H2S)
  resids <- pull(data, taste) - fitted
}

get_resids <- Vectorize(get_resids, c('slope', 'intercept'), SIMPLIFY = FALSE)
```

Now we can add a column to our `MLE_grid` with the *residuals* for each row. This will take a while to run. When it's done, look what has happened: each *row* of the dataset contains a *list* of 30 residuals! (Still totally cool.)

```
MLE_grid <- MLE_grid |>
  mutate(resids = get_resids(slope, intercept, data = cheddar))
glimpse(MLE_grid)
```

```
Rows: 10,291
Columns: 4
$ intercept <dbl> -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, ~
$ slope     <dbl> 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, ~
$ fitted    <list> <-5.595, 0.129, 1.314, 7.488, -3.579, 7.803, 11.178, 8.898,~
$ resids    <list> <17.895, 20.771, 37.686, 40.412, 9.179, 18.097, 26.122, 13.~
```

Estimate the residual standard deviation

To be able to get the **likelihood** from the **residuals**, we need to first estimate the residual standard deviation σ. The code below takes each list of 30 residuals and computes its standard deviation (`sd`).

```
MLE_grid <- MLE_grid |>
  mutate(sigma = sapply(resids, sd))
glimpse(MLE_grid)
```

```
Rows: 10,291
Columns: 5
$ intercept <dbl> -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, ~
$ slope     <dbl> 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, ~
$ fitted    <list> <-5.595, 0.129, 1.314, 7.488, -3.579, 7.803, 11.178, 8.898,~
$ resids    <list> <17.895, 20.771, 37.686, 40.412, 9.179, 18.097, 26.122, 13.~
$ sigma     <dbl> 12.17281, 12.07107, 11.97226, 11.87643, 11.78366, 11.69402, ~
```

Once you've gotten this far, check in with the prof to let her know where you are and if you have any questions.

Compute the actual likelihood

Finally, we are ready to compute the **(natural log of) the likelihood of the dataset, given each proposed model (each "proposed model" is one set of candidate β_0, β_1,** and σ).

For each row, we need to compute the **natural logarithm of the product of the likelihoods of all the residuals, in a normal distribution with mean 0 and sd σ.**

We will actually do the mathematical equivalent (which doesn't involve asking the computer to multiply together many very tiny numbers): find the **sum of the natural logarithms of the normal likelihoods of each residual value**.

Another explanation (add your own additional detail if needed):

- For each single residual, we find its likelihood by finding the density of a normal (mean = 0, standard deviation = σ) distribution for x = "our residual"
- We take the natural log of each residual's likelihood (working on a log scale reduces calculation problems for the computer)
- For each row of `MLE_grid` – that is, for each proposed slope/intercept combination – we add up all the residual log-likelihoods to get one joint log-likelihood for the whole dataset

```
get_loglik <- function(resids, sigma){
  LL <- sum(dnorm(resids, mean = 0, sd = sigma, log = TRUE))
}

get_loglik <- Vectorize(get_loglik, c('resids', 'sigma'), SIMPLIFY = TRUE)
```

Be patient, this will take a minute:

```
MLE_grid <- MLE_grid |>
  mutate(log_likelihood = get_loglik(resids, sigma))
```

```
glimpse(MLE_grid)
```

```
Rows: 10,291
Columns: 6
$ intercept      <dbl> -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, -15, ~
$ slope          <dbl> 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, ~
$ fitted         <list> <-5.595, 0.129, 1.314, 7.488, -3.579, 7.803, 11.178, 8~
$ resids         <list> <17.895, 20.771, 37.686, 40.412, 9.179, 18.097, 26.122~
$ sigma          <dbl> 12.17281, 12.07107, 11.97226, 11.87643, 11.78366, 11.69~
$ log_likelihood <dbl> -164.7479, -162.6843, -160.6096, -158.5268, -156.4392, ~
```

Best Model?

The BEST model (according to maximum likelihood estimation) is the one with the BIGGEST log-likelihood. Which one is that?

Use a plot to figure it out!

Note: Uncomment the code and run to see your plot, of course.

```
# gf_point(log_likelihood ~ slope, data = MLE_grid)
```

Why all the dots? Well, the intercept helps determine the likelihood, too…

```
# gf_point(log_likelihood ~ slope, data = MLE_grid,
#          color = ~intercept)
```

OK, *kind of* helpful – we can start to see where the highest values are…

Making it interactive might help.

```
library(plotly)
# gf_point(log_likelihood ~ slope, data = MLE_grid,
#          color = ~intercept) |>
#   ggplotly()
```

In the interactive plot, we can mouse over to find the exact slopes and intercepts that seem to have the highest likelihood.

To find the absolute top ones, we can also `arrange()` the dataset in order of `desc`ending likelihood and check out the top entries:

```
# MLE_grid |>
#   arrange(desc(log_likelihood)) |>
#   head()
```

Comparison

How do the slope and intercept that you just found compare with the one from `summary()` and `lm()`?

(If we did everything right, they should be quite close! We only estimated to one decimal place, though.)

Comments?

I am interested to know if your group found this exercise helpful – as a team or individually, let me know what was helpful/boring/confusing/awesome!

More time?

If you still have more time, fit a new model with **two** predictors for this dataset, and then do model assessment or selection (your choice) together. *Just for practice, of course, because of the "night-science-y" way we did model planning...*